11

Referenciando objetos

Transcripción

[00:00] Nos falta un comportamiento más que es transferir. Entonces nuevamente vamos a pensar como el usuario del banco. Cuando yo voy a transferir dinero, cuando yo entro a mi banca móvil, básicamente yo necesito saber a quién voy a transferir y también necesito especificarle cuánto le voy a transferir, entonces nuevamente venimos a nuestra clase cuenta y le vamos a crear el método transferir.

[00:31] Y en transferir vamos a decirle la cantidad double valor, de igual forma, el valor que vamos a transferir, y a que necesitamos dos parámetros, ya nos hemos dado cuenta que necesitamos dos parámetros, los necesitamos por coma. Con coma yo separo cuántos parámetros yo necesito para este método. Ahora. Dado que estamos trabajando ya a nivel de objetos, vamos a volver aquí, vamos a definir una segunda cuenta por ejemplo.

[01:10] Vamos a crear cuenta y cuentaDeJimena., esa va a ser otra cuenta, igual new cuenta. Perfecto. Aquí ya tenemos otra cuenta creada, tenemos miCuenta y la cuentaDeJimena. La ventaja de la orientación a objetos es que nos permite incluso tratar esta misma clase como objeto, porque propiamente es un objeto, y le vamos a decir que el segundo parámetro va a ser una cuenta, va a ser otra cuenta, y esta otra cuenta es el tipo cuenta.

[01:51] Es un objeto del tipo cuenta. Suena un poco confuso porque al igual que valor es un tipo double nosotros vamos a usar un objeto tipo cuenta, especificamos las llaves para el alcance y vamos a dejarlo como boolean,

vamos a dejar este método como boolean. Le damos public boolean, perfecto. Vemos aquí que ya se está saliendo un poco de la pantalla nuestro código.

[02:26] Para fines educativos vamos a darle un enter. Y él ya dividió automáticamente el método. Para ordenarlo vamos a darle tab, tabulaciones, y listo, ya tenemos digamos nuestros parámetros ordenados por un lado y nuestra ejecución del método va a ir aquí adentro nuevamente, ya que hemos definido la cuentaDeJimena y miCuenta, vamos a hacer la operación aquí.

[02:57] Vamos a decirle que ella tiene mucho dinero, le acaban de depositar 1000 soles. Perfecto. Ella ahora tiene 1000 soles en su cuenta, y ahora ella me va a transferir, buscamos aquí en los métodos, vemos que hay un montón de métodos, ya los vamos a explicar más adelante, pero nos interesa nuestro método transferir. Y aquí nos dice automáticamente especifica el valor.

[03:32] ¿Cuánto deseas transferir? Deseo transferir 400 soles. Y especifícanos la cuenta. ¿A qué cuenta deseas transferir? A miCuenta. Perfecto. Y vemos que podemos pasar también objetos como parámetros. El objeto miCuenta es una nueva referencia en la memoria entonces no significa que estamos aquí enviándole el objeto completo.

[04:03] Lo que esto significa es que le estamos enviando esta referencia en memoria para que él consiga obtener estas informaciones que ya están guardadas en este objeto. Recuerden ustedes que los objetos son todos independientes entre sí, pero ¿cómo podemos relacionarlos de esta forma? Yo estoy recibiendo la referencia al objeto cuenta de la memoria, por lo tanto yo tengo acceso a las informaciones en esta ubicación de memoria.

[04:35] Ahora vamos a implementar el método y nuevamente vamos a hacer la misma validación para ver si en efecto es factible hacerle la transferencia o no. Nuevamente if this.valor es mayor o igual que valor, perdón, this.saldo. Vimos ahí que no compila porque valor no existe en este objeto. Entonces, vamos a decirle que this.saldo cuando vamos a transferir, ¿qué significa?

[05:13] Que vamos a retirar dinero de nuestra cuenta para sumarlo en la otra cuenta. Entonces aquí le decimos que this.saldo va a ser igual a this.saldo menos valor, vamos a retirarle el valor solicitado y a la cuenta, al saldo de la cuenta que está especificada aquí, le vamos a aumentar ese valor. ¿Cómo hacemos eso? Decimos cuenta.saldo igual cuenta.saldo más valor.

[05:57] Pero yo ya tengo algo parecido en el objeto cuenta, que hace esta operación, que sería mi operación depositar. Yo tranquilamente podría llamar aquí cuenta depositar valor. Y el valor está definido aquí. Si todo sale éxito, al igual que hicimos aquí, voy a retornar true. Caso contrario, retorno falso. Listo. Vemos que el código ya está compilando y vamos a testar, vamos a probarlo.

[06:41] Y vamos a imprimir ahora, después que le he depositado y después que la cuentaDeJimena le hemos depositado 1000 soles y ella me ha transferido 400, vamos a hacer dos cosas. Vamos a ver cuánto saldo le queda a Jimena. Sería aquí, vamos a copiar dos veces. Aquí sería cuentaDeJimenta.saldo, y ahora cuánto saldo tengo en mi cuenta. Guardamos.

[07:15] Imprimimos y vemos en efecto que a ella le fueron retirados los 400 soles, ahora ya tiene 600, inicialmente tenía 1000, y a mí me fueron adicionados 400 soles. Vemos claramente ahora un poco más de la ventaja de orientar a objetos. Por ejemplo, a diferencia de las funciones normales, la que ahí sí especificamos por ejemplo para retirar, depositar, una función hablando proceduralmente sería "Deposítame a esta cuenta este valor".

[07:56] Pero una vez que está todo orientado a objetos ya tenemos digamos un sujeto del cual hablamos. Tenemos un personaje principal a quien hacemos referencia cuando definimos acciones para él. De esta forma el código también queda mucho más legible porque vemos cómo cuenta poco a poco ya no solamente sirve para guardar datos. Cuenta ya nos sirve para definir comportamientos propios que puede implementar el negocio.

[08:25] El código comienza a ser también más fácil de interpretar para una persona. Incluso sobre este código podemos hacer mejoras. ¿Cómo cuáles? Ya hemos aprendido que con más igual, nosotros aumentamos el valor de una misma variable, por lo tanto nosotros podíamos reemplazar esto tranquilamente así. A este saldo auméntale este valor. De igual manera aquí, a este saldo réstale este valor.

[09:05] Aquí por ejemplo, si ya hemos declarado o ya hemos explicado que return lo que hace es interrumpir la ejecución porque él ya retorna, él sale del método, este else ya no sería muy utilizado. Podríamos dejarlo tranquilamente como return false. ¿Por qué? Porque si él no retornó true, eso significa que esto no fue verdad, por lo tanto, si esto no se cumple, retorna falso.

[09:46] En la mayoría de veces de código Java van a ver esta estructura de código. Y ya que hemos visto también cómo podemos mejorar el código, hay un aspecto más que debemos tener en cuenta aquí: nosotros estamos ejecutando los métodos pero indiferentemente de que ellos no retornen o retornen valor. ¿Qué quiero decir? Que hasta ahora hay un método, que es del tipo void, que no retorna ningún valor, y hay otro método que retorna un booleano.

[10:25] Pero aquí no estamos viendo diferencia alguna. ¿Por qué? Vamos a verlo ahora. Por ejemplo, ¿en qué casos me sirve a mí un método que retorne valor? Cuando quizás yo necesite trabajar con el valor retornado. En este caso, si yo quisiera por ejemplo retornar un mensaje de éxito, si la transferencia es exitosa yo podría copiar todo esto de aquí, con la excepción de punto y coma, claro, aquí adentro.

[11:04] Decirle si este método de cuentaDeJimena transfiero 400 a miCuenta, entonces system.out.print transferencia exitosa. Perfecto. Ahora, ¿por qué yo estoy consiguiendo poner todo este método dentro de la sentencia if? Recordemos el curso pasado. La sentencia if recibe como único parámetro una condicional, una variable de tipo booleano, que puede ser verdad o falso.

[11:47] Entonces, sabiendo que este es un método que retorna valor si yo doy control y selecciono transferir, vemos que él es un método de tipo booleano. Por lo tanto él retorna o verdad o falso. ¿Qué quiere decir? Que él es totalmente elegible para entrar en esta condición. Si queremos optimizar aún más este código podemos simplemente escribir una condición, mejor dicho una variable de tipo booleano, boolean, puede transferir, y esa variable es igual a este método.

[12:34] Mejor dicho, no es igual a este método, es el resultado de ese método. Recuerden, este método va a retornar alguna cosa, esta operación va a retornar un booleano, el booleano es el resultado de esta operación. Por lo tanto, volvemos acá, por lo tanto, si puede transferir digamos aquí, si puede transferir, entonces transferencia exitosa. Si no, entonces no es posible. Perfecto.

[13:18] Vamos a ejecutar el código, guardamos todo, y vemos claramente que la transferencia fue exitosa.